UIC Fall 2020 ECE 464 Project 3: Are LFSRs good for covering multi-faults?

1st Trent Mathews
University of Illinois at Chicago
Department of ECE
Chicago, United States
tmathe24@uic.edu

2nd Ali Jafar
University of Illinois at Chicago
Department of ECE
Chicago, United States
ajafar5@uic.edu

3rd Steve Martell
University of Illinois at Chicago
Department of ECE
Chicago, United States
smarte7@uic.edu

***Abstract*—The goal of this project it to analyze the effectiveness of LFSRs at covering multi-faults.**

***Index Terms*—LFSR as PRPG, multi-faults and detection, Coverage of multi-faults**

I. Introduction and experiment design/setup

We start by defining what a multi-fault is. A multi-fault is a combination of two or more single faults that do not conflict nor share the same node. For example, a-0 and a-1 is not a valid multi-fault, but a-0 and b-0 would be valid. In order to determine the effectiveness of LFSRs at covering multi-faults, it is necessary to compare the outputs of a good circuit to the multi-fault circuit to see if the multi-faults are detected. If any of the primary output (PO) nodes have a different value in the multi-fault circuit, this is detection.

A. *Experiment Design*

To begin the experiment, generate a list of test vectors (TVs) using LFSR as generation method. For experiment purposes we only generate up to 100 TVs. This list of TVs will be used for input to the circuit during simulation. The next task is to produce a multi-fault list. For our experiment we produce all possible combination of two faults which do not conflict (for activation purposes) before the simulation begins. Finally simulate the circuit with all of the generated multi-faults with all of the generated TVs.

To form an analysis on coverage, we test all the multi-faults with every TV. This gives a list of detected multi-faults per TV. With this we can simulate a coverage analysis on the multi-fault list. With the results from the coverage analaysis we can make a conclusion on LFSRs for covering multi-faults.

B. *Experiment Design Setup*

The experiment setup consists of using Python to simulate a circuit. With the simulation results we can make a dictionary to keep track of all the mulit-faults that a single TV detects. The TVs are generated randomly using LFSR as a generation method. Specifically using an 8-bit LFSR, having taps at various configurations. Using simulation results for each multi-fault detected, we can create a coverage result for each TV.

Multi-Fault Coverage algorithm For a given circuit with its full multi-fault list F, given a list of test vectors (say $t_0, t_1, t_2, ..., t_7$):

```
• $T$ is the tv list, $F$ is the multi-fault list
• iterate $\forall t \in T$ :
    – $D = \emptyset$
    – iterate $\forall f \in F$ :
        apply $t$ to $f$:
        if $t$ detects $f$, $D = D + \{f\}$
    – report $t$ covers $D$
    – $F = F - D$
```

Using this algorithm conclusions can be made on the coverage for mulit-faults. The coverage for each 10 TV is grouped into one set. Our results will include batches of 10 TVs. Each batch is testing the same multi-faults with different TVs. This will ensure we are testing for detection of different faults, rather than detection of the same faults.

II. Python simulator functionalities, UI and Examples

In this project, we modified our Python fault simulator to handle multi-faults and generate a multi-fault list based on the circuit's full fault list. In order to accomplish this our program had the following components:

- A) generate a full multi-fault list and the ability to simulate any of them
- B) perform "batch" multi-fault simulation for a list of test vectors
- C) parse through a set of faults while tracking the fault coverage

A. *Full Multi-Fault List Generation*

Given a circuit bench file, produce the full multi-fault list. For our simulation purposes we generate a list of all possible two combinations of faults with no repeated or conflicting grouping. This will be the main list of possible multi-faults to use. The simulator can use all of them, or a random number of unique mulit-faults. This will give a reasonable data set to work with and simulate on. For example:

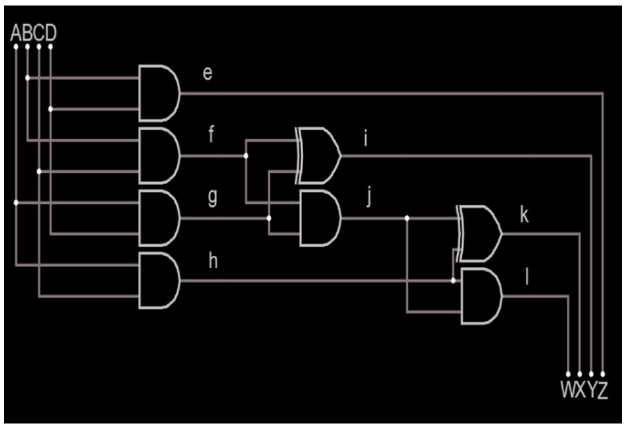


Fig. 1. Example of a 2-bit multiplier.

```
# circuit.bench
INPUT(a)
INPUT(b)
INPUT(c)
INPUT(d)
e = AND(b, d)
f = AND(b, c)
g = AND(a, d)
h = AND(a, c)
i = XOR(f, g)
j = AND(f, g)
k = XOR(h, j)
l = AND(h, j)
w = BUFF(l)
x = BUFF(k)
y = BUFF(i)
z = BUFF(e)
OUTPUT(w)
OUTPUT(x)
OUTPUT(y)
OUTPUT(z)
```

With this circuit the program generates all possible single-stuck-at faults (without repetition):

a-0, a-1, b-0, b-1, c-0, c-1, d-0, d-1, e-0, e-1, e-b-0, e-b-1, e-d-0, e-d-1, f-0, f-1, f-b-0, f-b-1, f-c-0, f-c-1, g-0, g-1, g-a-0, g-a-1, g-d-0, g-d-1, h-0, h-1, h-a-0, h-a-1, h-c-0, h-c-1, i-0, i-1, i-f-0, i-f-1, i-g-0, i-g-1, j-0, j-1, j-f-0, j-f-1, j-g-0, j-g-1, k-0, k-1, k-h-0, k-h-1, k-j-0, k-j-1, l-0, l-1, l-h-0, l-h-1, l-j-0, l-j-1, w-0, w-1, w-l-0, w-l-1, x-0, x-1, x-k-0, x-k-1, y-0, y-1, y-i-0, y-i-1, z-0, z-1, z-e-0, z-e-1, w-0, w-1, x-0, x-1, y-0, y-1, z-0, z-1 A total Fault count of 80 single faults.

Using this list we can generate all possible two combinations of multi-faults. This circuit produced a total of **2992** combination of multiple faults. This list can be entirely simulated on or partially simulated on. The simulator gives the option for using all of the list or choose random ones (without repeating) up to a number specified by the user. Example of using 100 randomly picked multi-fault for simulation:

['i-f-0', 'x-0'], ['i-f-1', 'w-1'], ['c-0', 'k-h-0'], ['g-d-1', 'z-e-1'], ['w-l-1', 'z-0'], ['k-h-1', 'l-h-1'], ['i-f-0', 'k-h-1'], ['e-d-0', 'f-b-1'], ['g-1', 'i-0'], ['d-1', 'z-e-1'], ['j-g-1', 'k-h-0'], ['y-1', 'z-0'], ['c-1', 'j-f-0'], ['i-1', 'z-0'], ['g-a-1', 'j-1'], ['i-g-0', 'x-0'], ['h-c-1', 'k-0'], ['f-1', 'j-0'], ['l-h-0', 'z-0'], ['x-1', 'y-0'], ['i-f-0', 'l-h-1'], ['h-1', 'k-1'], ['k-j-1', 'x-0'], ['y-1', 'z-1'], ['g-0', 'x-k-0'], ['e-b-0', 'z-0'], ['g-1', 'h-c-1'], ['l-0', 'y-i-1'], ['f-b-0', 'j-f-1'], ['j-f-0', 'y-i-0'], ['j-f-0', 'k-j-0'], ['j-g-0', 'k-h-0'], ['h-1', 'w-l-0'], ['w-1', 'z-1'], ['b-1', 'h-1'], ['k-1', 'k-j-1'], ['f-1', 'i-1'], ['h-1', 'l-j-1'], ['e-d-1', 'w-0'], ['h-a-0', 'z-1'], ['i-0', 'x-0'], ['g-0', 'y-i-0'], ['i-g-0', 'j-1'], ['i-g-0', 'x-k-0'], ['g-1', 'j-f-0'], ['j-0', 'j-g-1'], ['j-g-1', 'z-e-0'], ['c-0', 'l-h-1'], ['i-f-1', 'l-1'], ['h-a-0', 'y-i-0'], ['d-1', 'e-b-0'], ['b-0', 'h-0'], ['g-a-0', 'i-1'], ['f-c-1', 'l-1'], ['z-0', 'y-1'], ['f-c-0', 'g-d-0'], ['l-h-1', 'z-0'], ['e-b-1', 'l-0'], ['a-0', 'g-0'], ['g-a-1', 'y-0'], ['f-c-1', 'k-j-1'], ['g-a-1', 'y-i-0'], ['i-1', 'l-1'], ['a-1', 'x-0'], ['b-1', 'c-1'], ['k-1', 'w-0'], ['e-1', 'g-0'], ['j-g-1', 'z-1'], ['j-f-0', 'l-j-1'], ['f-b-0', 'y-0'], ['e-b-0', 'j-1'], ['k-j-0', 'z-0'], ['g-1', 'j-0'], ['c-1', 'z-e-0'], ['e-1', 'h-a-0'], ['i-f-1', 'l-h-1'], ['b-1', 'h-a-0'], ['w-1', 'x-k-1'], ['i-g-1', 'j-f-1'], ['d-0', 'w-l-0'], ['d-0', 'h-1'], ['e-1', 'g-d-1'], ['i-1', 'j-1'], ['j-1', 'x-0'], ['e-d-0', 'k-0'], ['y-1', 'x-0'], ['e-b-0', 'w-0'], ['g-d-0', 'y-0'], ['j-g-0', 'y-1'], ['i-0', 'k-h-1'], ['h-c-1', 'z-e-0'], ['l-1', 'z-e-0'], ['x-0', 'y-i-1'], ['d-0', 'j-g-0'], ['b-1', 'j-f-1'], ['f-b-0', 'i-0'], ['x-k-0', 'w-1'], ['f-b-0', 'l-j-0'], ['d-0', 'z-0'], ['h-1', 'i-0']

Using this list of multi-faults, the simulator will now begin testing for detection with all the multi-faults and all the TV that got generated. Keeping track of the multi-faults detected per TV.

B. *multi-fault simulation*

The simulator runs as normal for detection. It compares the output for a good circuit vs the output for a bad circuit. If there is a difference in the output, then the multi-fault is detected. The changes now are that it can handle multi-faults. The simulator groups the average of detection for each test vector into groups of 10. These batches consist of unique test vectors and the same multi-fault list. Counting the number of detected multi-faults.

Here is an example of coverage results for the circuit in Fig. 1. We will be using 100 randomly generated multi-faults, and 100 TVs generated via 8-bit LFSR with taps @2,3,4; that is $h_0 : h_7$ = 10111000.

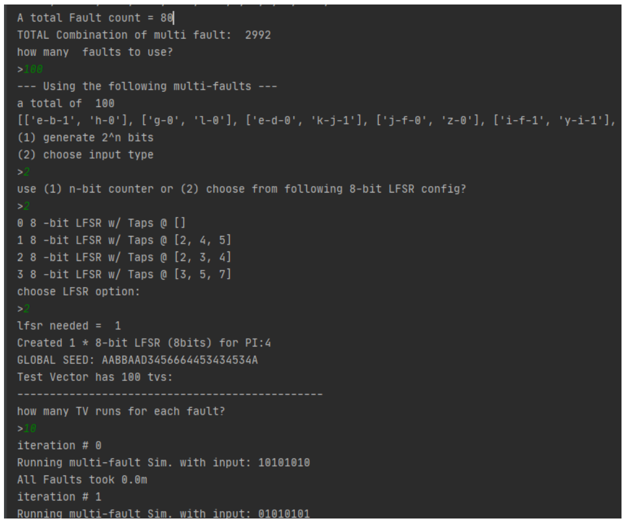
tv 10101010 detects the following 13 Faults: [['f-b-1', 'x-0'], ['a-1', 'w-1'], ['k-h-1', 'w-1'], ['e-b-1', 'z-1'], ['l-1', 'y-1'], ['a-1', 'y-1'], ['f-c-0', 'x-0'], ['e-0', 'w-1'], ['k-h-0', 'y-1'], ['j-g-1', 'y-1'], ['k-1', 'x-0'], ['e-d-0', 'z-1'], ['l-j-0', 'w-1']] with 87 Remaining Faults
tv 01010101 detects the following 10 Faults: [['j-f-0', 'z-0'], ['j-g-0', 'x-1'], ['i-g-0', 'z-0'], ['h-c-1', 'x-1'], ['x-1', 'y-i-1'], ['g-d-0', 'z-0'], ['z-e-1', 'x-1'], ['e-1', 'z-0'], ['d-1', 'x-1'], ['f-b-1', 'x-1']] with 77 Remaining Faults:

tv 10010010 detects the following 4 Faults: [['l-0', 'y-0'], ['h-a-0', 'y-0'], ['l-j-0', 'y-0'], ['i-g-1', 'y-0']] with 73 Remaining Faults:
tv 01001001 detects the following 0 Faults: [] with 73 Remaining Faults:
tv 10011100 detects the following 0 Faults: [] with 73 Remaining Faults:
tv 01001110 detects the following 0 Faults: [] with 73 Remaining Faults:
tv 00100111 detects the following 0 Faults: [] with 73 Remaining Faults:
tv 10101011 detects the following 0 Faults: [] with 73 Remaining Faults:
tv 11101101 detects the following 0 Faults: [] with 73 Remaining Faults:
tv 11001110 detects the following 0 Faults: [] with 73 Remaining Faults:
tv 01100111 detects the following 0 Faults: [] with 73 Remaining Faults:
tv 10001011 detects the following 0 Faults: [] with 73 Remaining Faults:
tv 11111101 detects the following 2 Faults: [['b-0', 'w-0'], ['h-c-1', 'w-0']] with 71 Remaining Faults:
tv 11000110 detects the following 0 Faults: [] with 71 Remaining Faults:
tv 01100011 detects the following 0 Faults: [] with 71 Remaining Faults:
tv 10001001 detects the following 0 Faults: [] with 71 Remaining Faults:
tv 11111100 detects the following 0 Faults: [] with 71 Remaining Faults:
tv 01111110 detects the following 0 Faults: [] with 71 Remaining Faults:
tv 00111111 detects the following 0 Faults: [] with 71 Remaining Faults:
tv 10100111 detects the following 0 Faults: [] with 71 Remaining Faults:

Note: only showing the first 20 coverage results

C. *coverage generation*

Each test vector runs and keeps track of which faults were detected as to eliminate double counting. After ten fault instances per test vector, the coverage is calculated from the number of unique multi-faults detected from the random 100 multi-faults chosen. 100 multi-faults were chosen due to computation and time restraints as simulating 100s of thousands of faults that were generated in larger circuits thus rendering using the full multi-fault list unfeasible for the project.



```
A total Fault count = 80
TOTAL Combination of multi fault:  2992
how many  faults to use?
>100
--- Using the following multi-faults ---
a total of  100
[['e-b-1', 'h-0'], ['g-0', 'l-0'], ['e-d-0', 'k-j-1'], ['j-f-0', 'z-0'], ['i-f-1', 'y-i-1'],
(1) generate 2^n bits
(2) choose input type
>2
use (1) n-bit counter or (2) choose from following 8-bit LFSR config?
>2
0 8 -bit LFSR w/ Taps @ []
1 8 -bit LFSR w/ Taps @ [2, 4, 5]
2 8 -bit LFSR w/ Taps @ [2, 3, 4]
3 8 -bit LFSR w/ Taps @ [3, 5, 7]
choose LFSR option:
>2
lfsr needed =  1
Created 1 * 8-bit LFSR (8bits) for PI:4
GLOBAL SEED: AABBAAD3456664453434534A
Test Vector has 100 tvs:
-----------------------------------------------
how many TV runs for each fault?
>10
iteration # 0
Running multi-fault Sim. with input: 10101010
All Faults took 0.0m
iteration # 1
Running multi-fault Sim. with input: 01010101
```

Fig. 2. Example of UI

III. Data and Results Presentation

We show the data and graphs for the different runs and the different circuits. All of the LFSR are generated with 8-bits and a global seed of **"AABBAAD3456664453434534A"**. Depending on the PI for a circuit, the seed will get extended with itself. This allows us to feed in the appropriate amount of bits to initialize the LFSR.

IV. Explanation, discussion, and conclusions.

The results that we have obtained paint a clear picture that LFSRs are not good for covering multi-faults. It was at first unclear what the results would show, but testing showed that combining every fault that was detected individually by a test vector into a multi-fault would also be yield a multi-fault that was detected. Similarly, the multi-faults that were generated by single faults that were undetected by a test vector were still undetected. Thus, the conclusion that we reached was that the explanation for the poor coverage was that only multi-faults generated from two or more detected single-faults could be detected. If one fault of a pair was undetected, the generated multi-fault would also be undetected. When looking at the the number of total faults compared to the number of detected faults per test vector, it is easy to see that when using the combination formula that each test vector would have a much smaller maximum multi-faults that can be detected compared to the true maximum. This was further made evident by the fault coverage dropping to 0 after the first ten faults as the likelihood of detecting more unique multi-faults that were also made of combinations of detected single faults was very low. In conclusion, we have found that LFSRs are not good at covering multi-faults in all but the simplest circuits.

V. Python simulation tool

Link to Github!

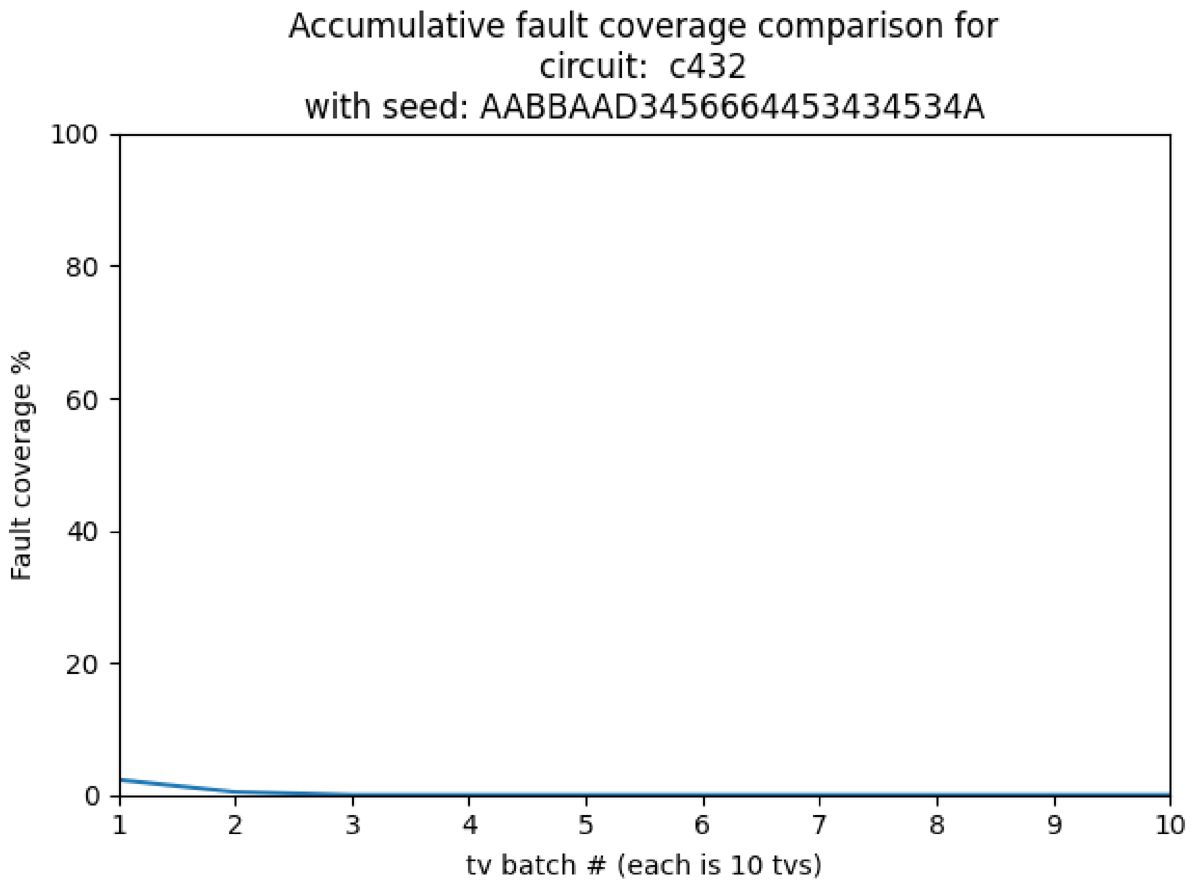


Fig. 3. c432 Coverage Results

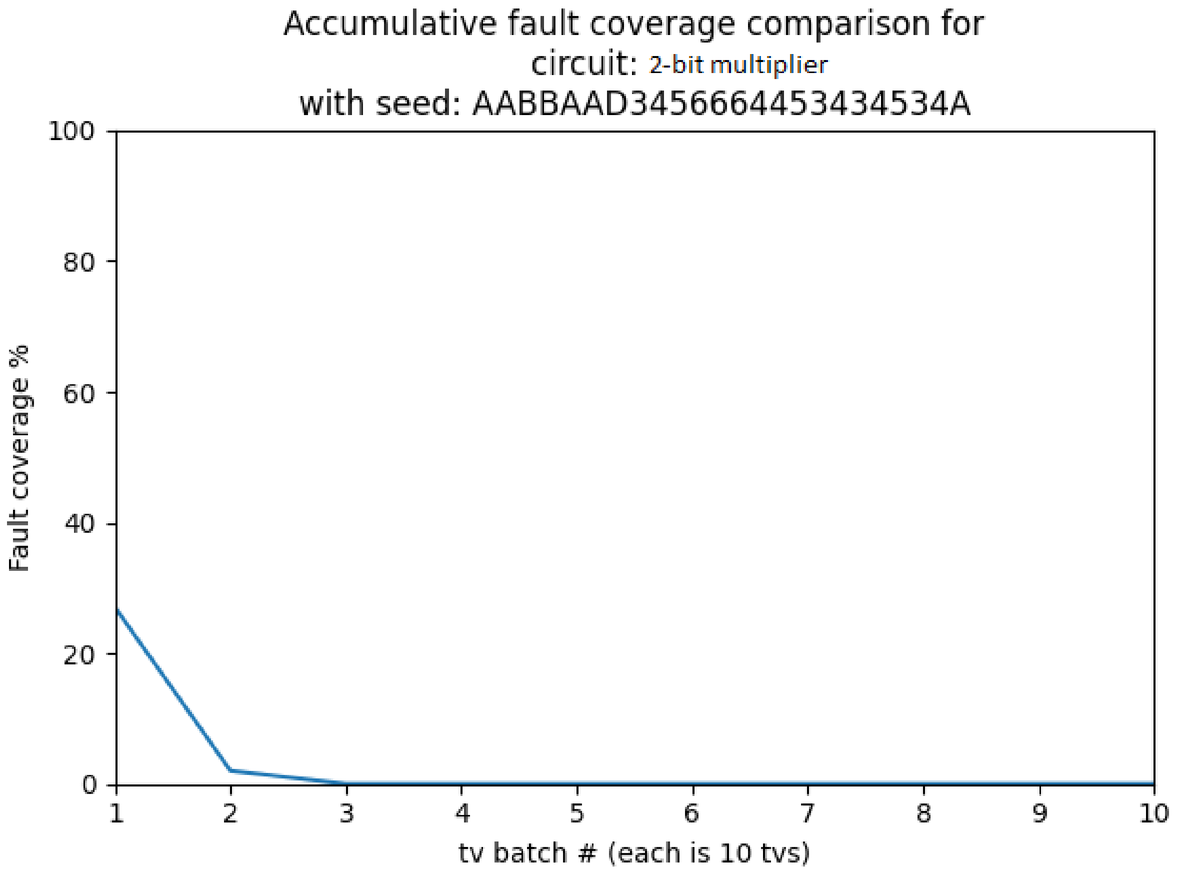


Fig. 4. 2-bit multiplier Coverage Results

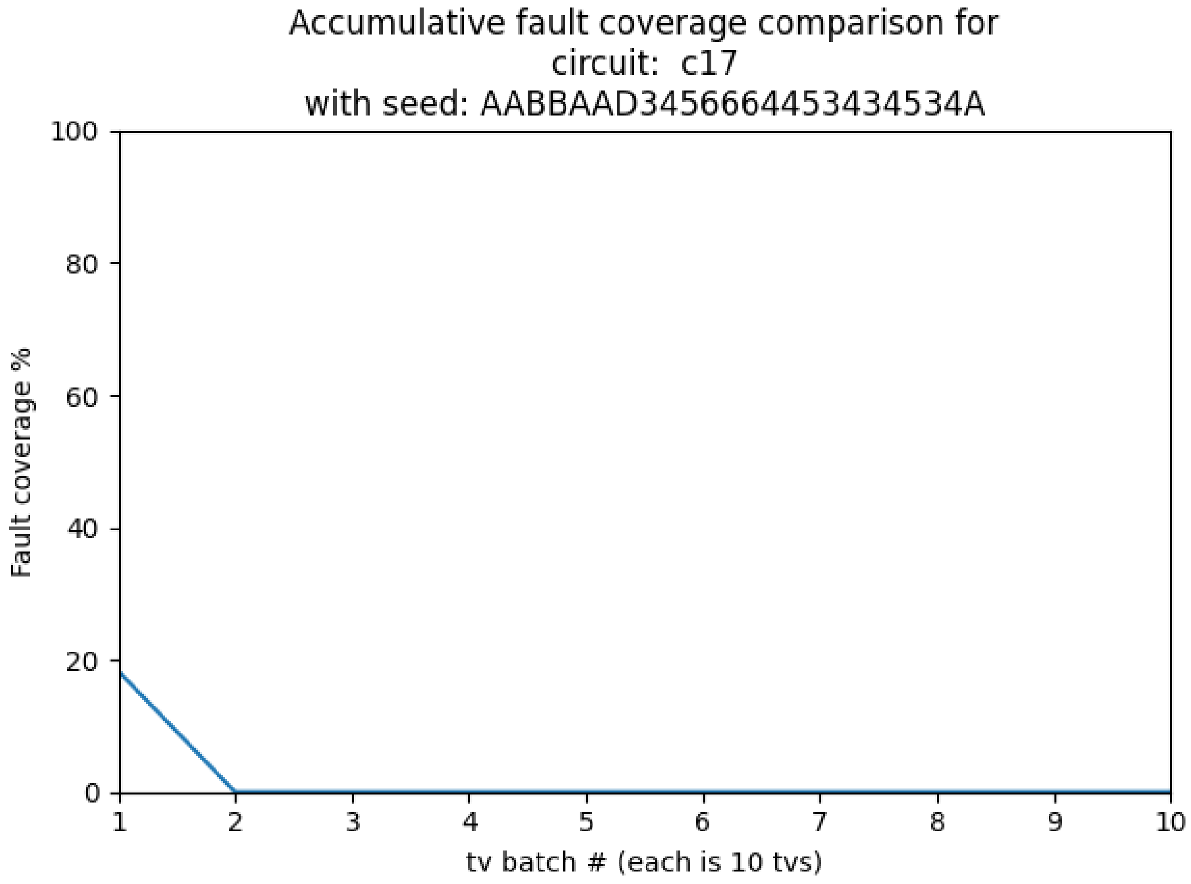


Fig. 5. c17 Coverage Results